

```
/* mod.c

Copyright (c) 1993-2012. Free Software Foundation, Inc.

This file is part of GNU MCSim.

GNU MCSim is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 3
of the License, or (at your option) any later version.

GNU MCSim is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with GNU MCSim; if not, see <http://www.gnu.org/licenses/>

-- Revisions -----
Logfile: %F%
Revision: %I%
Date: %G%
Modtime: %U%
Author: @a
-- SCCS -----

Entry point for model code generator.

Calls ReadModel() to define the model and WriteModel() to create
the output file.

The INPUTINFO struct is defined as follows:

wContext      :      A flag for the current context of the input.

pvmGloVars :      Vars of type ID_LOCAL* are accessible
                  only within the respective section. If
                  states are given a value outside of the
                  Dynamics section, it is used as an INITIAL
                  value, otherwise they are initialized to 0.0.

pvmDynEqns: List of equations to go into the CalcDeriv(),
pvmJacobEqns: Jacobian(), and ScaleModel() routines,
respectively.
pvmScaleEqns:      The LHS of equations of state variables in
Dynamics
                  are actually dt(). The hType field gives the type of
                  the LHS and also a flag for space in the uppermost
                  bit.
pvmCalcOutEqns: List of equations to into CalcOutputs(). These
can be used to scale output variables *only*.
The routinine is called just be outputting
values specified in Print().
```

```

/*
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include " getopt.h"
#include " lexerr.h"
#include " mod.h"
#include " modi.h"
#include " modiSBML.h"
#include " modo.h"
#include " strutil.h"

/* Globals */
static char vszOptions[] = "hHDRG";
static char vszFilenameDefault[] = "model.c";
char szFileWithExt[MAX_FILENAME_SIZE];

/* -----
-----
AnnounceProgram
*/
void AnnounceProgram (void)
{
    printf ("\n_____ \n");
    printf ("\nMod " VSZ_VERSION " - Model Generator for MCSim\n\n");

    printf ("MCSim and associated software comes with ABSOLUTELY NO
WARRANTY;\n\n");
    "This is free software, and you are welcome to redistribute
it\n";
    "under certain conditions; see the GNU General Public
License.\n\n");

#define HAVE_LIBSBML
    printf ("Using LibSBML.\n\n");
#endif

} /* AnnounceProgram */

/* -----
-----
PromptFilenames

prompts for both input and output file names. The space allocated
for inputting the files is reallocated to their actual size.
*/
void PromptFilenames (PSTR *pszFileIn, PSTR *pszFileOut)
{
    if (!(*pszFileIn = (PSTR) calloc (1, MAX_FILENAME_SIZE)))

```

```

ReportError (NULL, RE_OUTOFMEM | RE_FATAL, "PromptFilenames", NULL);

if (!(*pszFileOut = (PSTR) calloc (1, MAX_FILENAME_SIZE)))
    ReportError (NULL, RE_OUTOFMEM | RE_FATAL, "PromptFilenames", NULL);

printf ("Input filename? ");

fgets (*pszFileIn, MAX_FILENAME_SIZE, stdin);
*pszFileIn = strtok (*pszFileIn, " \t\n");

if (!(*pszFileIn)) /* Nothing entered, quit */
    return;

if ((*pszFileIn)[0]) { /* Input file specified */
    printf ("Output filename? ");

    fgets (*pszFileOut, MAX_FILENAME_SIZE, stdin);
    *pszFileOut = strtok (*pszFileOut, " \t\n");
}

if (!(*pszFileOut) || !(*pszFileOut)[0]) { /* If no output specified */
    free (*pszFileOut); /* .. use default later */
    *pszFileOut = NULL;
}
else {
    if (!(*pszFileIn = (PSTR) realloc (*pszFileIn, MyStrlen(*pszFileIn)
+ 1)))
        ReportError (NULL, RE_OUTOFMEM | RE_FATAL, "PromptFilenames",
NULL);
    if (!(*pszFileOut = (PSTR) realloc (*pszFileOut,
MyStrlen(*pszFileOut) + 1)))
        ReportError (NULL, RE_OUTOFMEM | RE_FATAL, "PromptFilenames",
NULL);
}

} /* PromptFilenames */

/*
-----
ShowHelp
*/
void ShowHelp ()
{
    printf ("Help:\n");
    printf ("Usage: mod [options] [input-file [output-file]]\n");
    printf ("Options:\n");
    printf ("  -h  Display this information\n");
    printf ("  -H  Display this information\n");
    printf ("  -D  Debug mode\n");
    printf ("  -R  Generate an R deSolve compatible C file\n");

    printf ("Creates file 'output-file' (or 'model.c', by default)\n");
    printf ("according to the input-file specifications.\n\n");
}

```

```

} /* ShowHelp */

/*
-----  

-----  

GetCmdLineArgs  

    Retrieves options and filenames from the command line arguments passed  

to  

the program.  

The command line syntax is:  

    mod [options] [input-file [output-file]]  

If the output filename is not given a default (model.c) is used.  

Options with arguments can be used to give a collection of input  

files to process and merge into a single output file.  

If neither the input, nor output filenames are given, the  

program prompts for them both.  

The options can appear anywhere in the line and in any order.  

The options are parsed with _getopt(). After _getopt() is called,  

the args in rgszArg have been permuted so that non-option args are  

first, which in this case means the input and output filenames.  

Uses the following globals:  

    char * optarg;      -- Contains the string argument to each option in  

turn  

    int    optind;       -- Index in ARGV of the next elem to be scanned  

    char * nextchar;    -- The next char to be scanned in the option-  

element  

    int    opterr;       -- 0 value flags to inhibit GNU error messages  

*/  

void GetCmdLineArgs (int nArg, char *const *rgszArg, PSTR *pszFileIn,  

                    PSTR *pszFileOut, PINPUTINFO pinfo)  

{  

    int c;  

    opterr = 1; /* inhibit internal -getopt error messages */  

    *pszFileIn = *pszFileOut = (PSTR) NULL;  

    while (1) {  

        c = _getopt (nArg, rgszArg, vszOptions);  

        if (c == EOF) /* Finish with option args */  

            break;  

        switch (c) {

```

```

case 'D':
    /* Could setup to run with certain debug flags, not used */
    printf (">> Debug mode using option '%s': "
           "Not implemented, ignored.\n\n", optarg);
    break;

case 'H':
case 'h':
    ShowHelp ();
    exit (0);
    break;

case 'R':
    printf (">> Generating code for linking with R deSolve
package.\n\n");
    pinfo->bforR = TRUE;
    break;

case '?':
default:
    ShowHelp ();
    exit (0);

} /* switch */

} /* while */

switch (nArg - optind) { /* Remaining args should be filenames */

case 2: /* Output and input file specified */
    *pszFileOut = rgszArg[optind + 1];

    /* Fall through! */

case 1: /* Input file specified */
    *pszFileIn = rgszArg[optind];
    break;

case 0: /* No file names specified */
    PromptFilenames (pszFileIn, pszFileOut);
    break;

default:
    printf ("mod: too many parameters on command line\n");
    ShowHelp ();
    exit (-1);
    break;

} /* switch */

while (*pszFileIn && (*pszFileIn)[0] && /* Files specified */
      !MyStrcmp(*pszFileIn, *pszFileOut)) { /* and not different */

    printf ("\n** Input and output filename must be different.\n");

```

```

    PromptFilenames (pszFileIn, pszFileOut);
} /* while */

if (!(*pszFileIn && (*pszFileIn)[0])) { /* no input name given is an
error */
    printf ("Error: an input file name must be specified - Exiting\n\n");
    exit (-1);
}

/* store input file name for use by modo.c */
pinfo->szInputFilename = *pszFileIn;

/* use default output file name if it is missing */
if (!*pszFileOut)
    *pszFileOut = vszFilenameDefault;

} /* GetCmdLineArgs */

/*
-----
-----
    InitInfo
*/
void InitInfo (PINPUTINFO pinfo, PSTR szModGenName)
{
    pinfo->wContext      = CN_GLOBAL;
    pinfo->bTemplateInUse = FALSE;
    pinfo->bforR          = FALSE;
    pinfo->szModGenName   = szModGenName;

    pinfo->pvmGloVars     = NULL;
    pinfo->pvmDynEqns     = NULL;
    pinfo->pvmScaleEqns   = NULL;
    pinfo->pvmJacobEqns   = NULL;
    pinfo->pvmCalcOutEqns = NULL;
    pinfo->pvmEventEqns   = NULL;
    pinfo->pvmRootEqns    = NULL;

    pinfo->pvmCpts        = NULL;
    pinfo->pvmLocalCpts   = NULL;

} /* InitInfo */

/*
-----
-----
    main -- Entry point for the simulation model preprocessor
*/
int main (int nArg, PSTR rgszArg[])
{
    INPUTINFO info;

```

```

INPUTINFO tempinfo;
PSTR szFileIn, szFileOut;

AnnounceProgram ();

#ifndef WIN32
/* replace '\' in rgszArg[0] with '/' */
{ int i;
  for (i = 0; i < strlen(rgszArg[0]); i++) {
    if (rgszArg[0][i] == '\\') rgszArg[0][i] = '/';
  }
}
#endif

InitInfo (&info, rgszArg[0]);
InitInfo (&tempinfo, rgszArg[0]);

GetCmdLineArgs (nArg, rgszArg, &szFileIn, &szFileOut, &info);

ReadModel (&info, &tempinfo, szFileIn);

/* I think that here we should manipulate info if a pure template has
   been read, assuming we care about that case, otherwise it should be
   an error to define a pure template without SBML to follow */

if (info.bforR == TRUE)
  Write_R_Model (&info, szFileOut);
else
  WriteModel (&info, szFileOut);

return 0;

} /* main */

```